2024

DÉTECTION D'ATTAQUES DDOS AVEC PYTORCH

AUTEUR :
JOSHUA JUSTE EMMANUEL YUN PEI NIKIEMA

PROFESSEUR : Dr. SY

Table des matières

Introduction

L'avènement des réseaux informatiques de plus en plus complexes et interconnectés a ouvert la porte à une multitude de cybermenaces, dont les attaques par déni de service distribué (DDoS) figurent parmi les plus redoutables. Ces attaques visent à submerger un réseau ou un serveur de requêtes illégitimes, le rendant ainsi indisponible aux utilisateurs légitimes.

Face à cette menace croissante, l'intelligence artificielle (IA) se présente comme un outil prometteur pour la détection et la mitigation des attaques DDoS. En effet, les algorithmes d'apprentissage automatique peuvent analyser de grands volumes de données réseau et identifier des modèles subtils indiquant une activité malveillante.

Dans ce contexte, ce projet s'inscrit dans l'exploration du potentiel de l'apprentissage profond pour la détection d'intrusions DDoS. En utilisant le Framework PyTorch, nous développerons un modèle de Deep Learning capable d'analyser le dataset IDS 2018 Intrusion CSVs (CSE-CIC-IDS2018) et de distinguer les trafics légitimes des trafics malveillants.

I. Contexte

Les attaques DDoS représentent un défi majeur pour la sécurité des réseaux informatiques. Elles peuvent causer des dommages importants aux entreprises et aux organisations en perturbant leurs opérations, en compromettant la disponibilité des services et en portant atteinte à leur réputation.

Les méthodes traditionnelles de détection des intrusions DDoS, basées sur des règles et des signatures, se révèlent souvent inefficaces face aux attaques sophistiquées et évolutives. En effet, ces méthodes ne parviennent pas à s'adapter aux nouveaux modes opératoires des attaquants et génèrent souvent de nombreux faux positifs, entraînant des perturbations inutiles du réseau.

L'apprentissage profond offre une approche alternative prometteuse pour la détection d'intrusions DDoS. En apprenant à partir de grands volumes de données réseaux, les modèles de Deep Learning peuvent identifier des patterns subtils et complexes qui indiquent une activité malveillante. Cette approche permet de surmonter les limitations des méthodes traditionnelles et d'améliorer la précision et l'efficacité de la détection des intrusions DDoS.

Ce projet vise à démontrer le potentiel de l'apprentissage profond pour la détection d'intrusions DDoS en développant un modèle de Deep Learning capable d'analyser le dataset IDS 2018 Intrusion CSVs (CSE-CIC-IDS2018) et de distinguer les trafics légitimes des trafics malveillants. Les résultats de ce projet permettront d'évaluer l'efficacité de l'apprentissage profond dans la lutte contre les attaques DDoS et d'ouvrir la voie à de nouvelles solutions de sécurité réseau plus robustes et performantes.

II. Présentation du Dataset

Le dataset IDS 2018 Intrusion CSVs (CSE-CIC-IDS2018) est un ensemble de données volumineux et public destiné à l'entraînement et à l'évaluation de modèles d'apprentissage automatique pour la détection d'intrusions. Créé par le Canadian Institute for Cybersecurity (CIC), il rassemble des données de trafic réseau étiquetées comme bénignes ou malveillantes.

L'ensemble de données lui-même était basé sur les journaux des serveurs de l'université, qui ont révélé diverses attaques DDoS tout au long de la période accessible au public. Lorsque vous rédigez des carnets d'apprentissage automatique pour ces données, notez que la colonne Label est sans doute la partie la plus importante des données, car elle détermine si les paquets envoyés sont malveillants ou non.

1. Caractéristiques principales du dataset

- Source : Récolté sur un réseau universitaire durant 10 jours en 2018.
- Taille : Plus de 16 millions d'enregistrements, chacun représentant un flux réseau.
- Attributs : Chaque enregistrement contient 80 attributs, incluant des statistiques réseau, des caractéristiques de flux et des informations sur les protocoles applicatifs.

  Les colonnes les plus importantes de cet ensemble de données sont énumérées ci-dessous.

  - Dst Port (Port de destination)
  - Protocole
  - Flow Duration
  - Tot Fwd Pkts (Total des paquets aller)
  - Tot Bwd Pkts (Total des paquets de retour)
  - Label (Label)
- Classes : Le dataset contient 81 classes, une pour chaque type d'attaque ou de trafic bénin.

2. Utilité du dataset IDS 2018 Intrusion CSVs

Le dataset IDS 2018 Intrusion CSVs constitue une ressource précieuse pour les chercheurs et praticiens développant des systèmes de détection d'intrusions. Sa structure claire et son étiquetage précis permettent l'entraînement de divers algorithmes d'apprentissage automatique.

3. Applications spécifiques du dataset

Entraînement de modèles de détection d'intrusions : Le dataset peut servir à entraîner des modèles d'apprentissage automatique pour identifier le trafic réseau malveillant.

Évaluation de modèles de détection d'intrusions : Le dataset permet d'évaluer la performance des modèles de détection d'intrusions.

Développement de nouvelles techniques de détection d'intrusions : Le dataset peut être utilisé pour développer de nouvelles techniques de détection d'intrusions, telles que l'extraction de caractéristiques et les algorithmes de détection d'anomalies.

En résumé, le dataset IDS 2018 Intrusion CSVs est un outil précieux pour quiconque s'intéresse au développement ou à l'utilisation de l'apprentissage automatique pour la détection d'intrusions.

4. Points supplémentaires à considérer

Le dataset est disponible gratuitement en format CSV.

Il est important de noter que le dataset contient des données simulées et ne reflète peut-être pas parfaitement les attaques DDoS réelles.

De nombreux articles de recherche ont été publiés utilisant le dataset IDS 2018 Intrusion CSVs, ce qui en fait une référence bien établie dans le domaine de la détection d'intrusions basée sur l'apprentissage automatique.

III. Etapes d'implémentation du modèle

Pour mener à bien la détection d'intrusions DDoS à l'aide de l'apprentissage profond, nous suivrons une méthodologie rigoureuse en plusieurs étapes clés.

1. Chargement des fichiers

Le dataset étant constitué de plusieurs fichiers (10), nous avons écrit une fonction capable de charger ces fichiers et ensuite les stocker dans un dataframe. Mais contenu de la capacité limitée de notre environnement nous n'avons pu charger que 7 fichiers.

```python
import psutil

def load_csv_files_progressively(directory, memory_threshold=70):
    """
    Charge tous les fichiers CSV dans un répertoire donné et les combine progressivement en un unique DataFrame.

    Args:
    - directory (str): Chemin du répertoire contenant les fichiers CSV.
    - memory_threshold (int): Seuil d'utilisation de la mémoire RAM en pourcentage pour arrêter la concaténation.

    Returns:
    - DataFrame: DataFrame combinant toutes les données des fichiers CSV progressivement.
    - int: Nombre de DataFrames concaténés.
    """

    # Liste pour stocker les DataFrames combinés progressivement
    combined_dfs = []
    file_to_exclude = ['02-20-2018.csv', '03-01-2018.csv', '02-16-2018.csv']

    # Parcours des fichiers dans le répertoire
    files = [file for file in os.listdir(directory) if file.endswith('.csv') and file not in file_to_exclude]


    # Surveillance de l'utilisation de la mémoire
    mem = psutil.virtual_memory()
```

Le mécanisme de chargement des fichiers CSV repose sur un principe simple : le chargement progressif et la surveillance de la mémoire RAM.

Plutôt que de charger tous les fichiers CSV en une seule fois, la fonction les traite un par un, ce qui permet de réduire considérablement la consommation de mémoire instantanée. De plus, elle surveille en permanence l'utilisation de la mémoire RAM et interrompt le processus de concaténation si le seuil défini est atteint. Cette approche permet d'éviter les dépassements de mémoire et les erreurs liées à l'insuffisance de RAM, tout en garantissant un traitement efficace des ensembles de données volumineux.

En résumé, le principe clé réside dans la combinaison du chargement progressif et de la gestion attentive de la mémoire RAM, permettant de transformer efficacement des fichiers CSV volumineux en Dataframe utilisables. Avec ce principe nous sommes arrivé à charger 9 fichiers mais pour la suite du projet la RAM se saturait.

2. Prétraitement des données

Cette étape cruciale consiste à préparer le dataset IDS 2018 Intrusion CSVs pour l'apprentissage par le modèle. Elle comprend des tâches telles que le nettoyage des données, la gestion des valeurs manquantes et la normalisation des attributs.

Pour cela nous avons implémenté deux (02) classes principales : Data Processor et DataLoaderManager. La première est celle qui fait le prétraitement des données à savoir le remplacement des valeurs manquantes, la mise à l'échelle et la conversion des colonnes en type numérique car lors du chargement, certains fichiers csv contiennent des éléments non numériques ou des valeurs manques ce qui faute la détermination des bons types des colonnes de ces fichiers. Par conséquent, pandas opte pour le type par défaut qui est `object`.

Voici le warning :

```
Fichier utilisé :  02-28-2018.csv
/tmp/ipykernel_34/216919516.py:31: DtypeWarning: Columns (0,1,3,4,5,6,7,8,9,1
0,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,
36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,6
1,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78) have mixed types. Speci
fy dtype option on import or set low_memory=False.
  df = pd.read_csv(file_path)
```

La deuxième classe a pour but de rendre les données chargeables pour l'entrainement et l'évaluation du modèle.

```python
class DataProcessor:
    """Classe pour prétraiter les données."""

    def __init__(self, device):
        """Initialiseur de la classe DataProcessor."""
        self.encoder = LabelEncoder()
        self.preprocessing_pipeline = Pipeline(steps=[
            ('imputer', SimpleImputer()),  # Remplace les valeurs manquantes
            ('scaler', StandardScaler())   # Met à l'échelle les caractéristiques
        ])
        self.device = device


    def preprocess_data(self, data, target_name):
        """Prétraite les données en remplissant les valeurs manquantes, en mettant à l'échelle et en encodant la variable cible.

        Args:
            data (pandas.DataFrame): Le DataFrame contenant les données à prétraiter.
            target_name (str): Le nom de la colonne contenant la variable cible.

        Returns:
            torch.Tensor, torch.Tensor: Les données prétraitées (caractéristiques) et la variable cible encodée.
        """

        target = data[target_name]
```

```python
class DataLoaderManager:
    """
    Classe pour gérer la création et la configuration des chargeurs de données pour l'entraînement et l'évaluation de modèles.
    """

    def __init__(self, batch_size_train=100, batch_size_test=20, device=None):
        """
        Initialise la classe DataLoaderManager.

        Args:
            batch_size_train (int, optional): Taille des lots pour l'entraînement. Par défaut, 100.
            batch_size_test (int, optional): Taille des lots pour l'évaluation. Par défaut, 20.
            device (torch.device, optional): Périphérique utilisé pour le calcul (CPU ou GPU).
                                             Si None, le périphérique sera déterminé automatiquement.
        """

        self.batch_size_train = 100
        self.batch_size_test = 20
        self.device = device # Détermine le périphérique

    def create_datasets_and_loaders(self, X_train, X_test, y_train, y_test):
        """
        Crée des ensembles de données et des chargeurs de données pour l'entraînement et l'évaluation.

        Args:
            X_train (np.ndarray): Données d'entraînement (caractéristiques).
            X_test (np.ndarray): Données de test (caractéristiques).
            y_train (np.ndarray): Étiquettes d'entraînement
```

3. Construction du modèle de Deep Learning

Nous concevrons et architecturerons un modèle de Deep Learning adapté à la tâche de détection d'intrusions DDoS. Le choix de l'architecture du réseau neuronal, de ses hyperparamètres et de ses fonctions d'activation sera effectué de manière méthodique en s'appuyant sur des connaissances théoriques et des expérimentations.

```python
class IntrusionDetectionModel(torch.nn.Module):
    """Classe pour le modèle de détection d'intrusion."""

    def __init__(self, in_features, out_features, device):
        """Initialiseur de la classe IntrusionDetectionModel."""
        super().__init__()
        self.fc1 = torch.nn.Linear(in_features, 50)
        self.fc2 = torch.nn.Linear(50, 100)
        self.fc3 = torch.nn.Linear(100, out_features)
        self.device = device
        self.to(self.device)

    def forward(self, x):
        """Méthode de propagation avant."""
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Le code Python définit la classe IntrusionDetectionModel qui encapsule l'architecture et le comportement du modèle.

Architecture du modèle :

Le modèle se compose de trois couches entièrement connectées (fc1, fc2, et fc3) définies par la classe torch.nn.Linear. Ces couches permettent au modèle d'apprendre des relations non-linéaires complexes entre les caractéristiques du trafic réseau (données d'entrée) et la classification en trafic légitime ou malveillant (sortie).

- **Couche d'entrée (**fc1**) :** Cette couche reçoit un vecteur d'attributs représentant un flux réseau issu du dataset IDS 2018 Intrusion CSVs. Le nombre d'unités d'entrée correspond au nombre de caractéristiques du dataset (indiqué par in_features dans le code). La couche projette ensuite l'entrée dans un espace de dimensionnalité plus faible (50 unités dans ce cas).
- **Couches cachées (**fc2**) :** Le modèle possède une couche cachée intermédiaire (fc2) composée de 100 unités. Cette couche joue un rôle crucial dans l'apprentissage de représentations abstraites des données d'entrée. La fonction d'activation ReLU (Rectified Linear Unit) est appliquée après chaque couche cachée pour introduire de la non-linéarité dans le modèle et permettre l'apprentissage de modèles complexes.

- **Couche de sortie (**fc3**) :** La dernière couche entièrement connectée (fc3) possède un nombre d'unités égal au nombre de classes de sorties (out_features). Dans le contexte de la détection d'intrusions DDoS, la couche de sortie aura typiquement deux unités, une pour le trafic légitime et une pour le trafic malveillant.

Fonctionnement du modèle :

La méthode forward de la classe IntrusionDetectionModel définit le processus de propagation avant du réseau. Pour un échantillon d'entrée x représentant un flux réseau, la méthode applique les opérations suivantes :

- **Propagation à travers la couche d'entrée (**fc1**) :** L'échantillon d'entrée x est multiplié par les poids de la couche fc1 et ajouté à un terme de biais. La fonction d'activation ReLU est ensuite appliquée à la sortie linéaire pour introduire de la non-linéarité.
- **Propagation à travers la couche cachée (**fc2**) :** La sortie de la couche fc1 est utilisée comme entrée pour la couche cachée fc2. Le même processus de multiplication matricielle, ajout de biais et application de la fonction ReLU est répété.
- **Propagation à travers la couche de sortie (**fc3**) :** La sortie de la couche cachée fc2 est multipliée par les poids de la couche de sortie fc3 et ajoutée à un terme de biais. Cependant, aucune fonction d'activation n'est appliquée à la couche de sortie car on souhaite obtenir des scores bruts représentant les probabilités non normalisées d'appartenance à chaque classe (trafic légitime ou malveillant).

Compilation du modèle :

Le modèle est compilé sur un périphérique de calcul spécifique (CPU ou GPU) à l'aide de la méthode to(self.device). Cela permet d'accélérer les calculs et d'améliorer les performances du modèle pendant l'entraînement et l'inférence.

4. Entraînement du modèle

Le modèle de Deep Learning sera ensuite entraîné sur le dataset préparé. Cette étape implique l'optimisation des paramètres du modèle afin de minimiser une fonction de perte, permettant au modèle d'apprendre à distinguer efficacement les trafics légitimes des trafics malveillants.

Pour cela nous avons implémenter une fonction pour la boucle d'entrainement.

```python
def train_model(model, criterion, optimizer, trainloader, max_iter, writer, device):
    """
    Fonction pour entraîner le modèle et enregistrer les pertes dans un fichier.

    Args:
        model (torch.nn.Module): Le modèle à entraîner.
        criterion (torch.nn.Module): La fonction de perte à utiliser pour calculer la perte.
        optimizer (torch.optim.Optimizer): L'optimiseur à utiliser pour mettre à jour les poids du modèle.
        trainloader (torch.utils.data.DataLoader): Le chargeur de données pour les données d'entraînement.
        max_iter (int): Le nombre total d'itérations d'entraînement (epochs).
        writer (SummaryWriter): L'écrivain Tensorboard pour enregistrer les statistiques d'entraînement.
        device (torch.device): Le périphérique d'exécution (GPU ou CPU).

    Returns:
        list: Une liste contenant les pertes moyennes à chaque époque.
    """

    model.to(device)  # Transférer le modèle sur le GPU s'il est disponible
    model.train()  # Mettre le modèle en mode d'entraînement

    losses = []  # Liste pour stocker les pertes moyennes à chaque époque

    for epoch in range(max_iter):  # Boucle sur le nombre total d'itérations d'entraînement (epochs)
        running_loss = 0  # Initialiser la perte courante à zéro

        for i, (data, target) in enumerate(trainloader):  # Boucle sur les données d'entraînement
```

5. Évaluation du modèle

La performance du modèle entraîné sera évaluée rigoureusement sur un ensemble de données de test indépendant. Divers indicateurs de performance ont été calculés pour mesurer l'efficacité du modèle dans la détection des intrusions DDoS, que sont la précision, l'accuracy, le rappel (Recall) et le F1-score.

```python
import torchmetrics as tm
from sklearn.metrics import precision_recall_fscore_support

def evaluate_model(model, criterion, testloader, device, Nb_class):
    """
    Fonction pour évaluer le modèle de détection d'intrusion sur l'ensemble de données de test et
    enregistrer les pertes individuelles et les métriques de performance.

    Args:
        model (torch.nn.Module): Le modèle de détection d'intrusion entraîné.
        criterion (torch.nn.Module): La fonction de perte utilisée pour l'évaluation.
        testloader (torch.utils.data.DataLoader): Le chargeur de données pour l'ensemble de données de test.
        device (torch.device): Le périphérique d'exécution (GPU ou CPU).

    Returns:
        dict: Un dictionnaire contenant les métriques de performance (perte, précision, rappel, F1-score, AUC).
    """

    model.eval()  # Basculer le modèle en mode évaluation

    losses = []  # Liste pour stocker les pertes individuelles
    precision_scores = []
    recall_scores = []
    f1_scores = []
    accuracies = []
```

IV. Analyse des résultats

Après l'évaluation de notre modèle sur les données de tests, nous avons eu les résultats suivants :

```
# Nombre de classes
Nb_class = len(df['Label'].unique())

# Évaluer le modèle sur l'ensemble de données de test et afficher les métriques
metrics = evaluate_model(model, criterion, testloader, device, Nb_class)

print("Métriques d'évaluation:")
for metric_name, value in metrics.items():
    print(f"{metric_name}: {value:.4f}")
print("Modèle évalué sur l'ensemble de données de test.")

# Fermer le writer de Tensorboard
writer.close()
print("Writer de Tensorboard fermé.")
```

```
Métriques d'évaluation:
Test set loss: 0.0526
Accuracy means: 0.9884
Precision: 0.9491
Recall: 0.9525
F1-score: 0.9506
Modèle évalué sur l'ensemble de données de test.
Writer de Tensorboard fermé.
```

D'après les métriques d'évaluation obtenues, le modèle de détection d'intrusions DDoS semble présenter de bonnes performances globales. Voici une analyse détaillée de chaque métrique.

1. Précision (Precision) : 0.9491

La précision indique la proportion de flux réseau correctement identifiés comme légitimes ou malveillants parmi les flux identifiés comme malveillants par le modèle. Une valeur élevée de précision (proche de 1) signifie que le modèle est peu susceptible de générer de faux positifs, c'est-à-dire de classer des flux légitimes comme malveillants. Dans ce cas, la précision de 0,9491 indique que le modèle est précis à 94,91 % dans l'identification des flux malveillants.

2. Rappel (Recall) : 0.9525

Le rappel indique la proportion de flux réseau malveillants réellement détectés par le modèle parmi tous les flux malveillants présents dans l'ensemble de données de test. Une valeur élevée de rappel (proche de 1) signifie que le modèle est peu susceptible de générer de faux négatifs, c'est-à-dire de manquer des flux malveillants. Ici, le rappel de 0,9525 suggère que le modèle détecte 95,25 % des flux malveillants.

3. F1-score : 0.9506

Le F1-score est une mesure composite qui prend en compte à la fois la précision et le rappel. Il est calculé comme la moyenne harmonique de la précision et du rappel. Un F1-score élevé (proche de 1) indique que le modèle offre un bon équilibre entre la précision et le rappel. Dans ce cas, le F1-score de 0,9506 confirme que le modèle présente un bon compromis entre la capacité à identifier correctement les flux légitimes et à détecter efficacement les flux malveillants.

4. Accuracy (Exactitude) : 0.9884

L'exactitude globale de 0,9884 indique que le modèle classifie correctement 98,84 % des flux réseau, qu'ils soient légitimes ou malveillants. Ce résultat élevé confirme la performance globale du modèle dans la distinction entre les trafics légitimes et les attaques DDoS.

5. Interprétation globale

En considérant l'ensemble des métriques, on peut conclure que le modèle de détection d'intrusions DDoS présente des performances prometteuses. Il offre une précision élevée dans l'identification des flux malveillants (faible taux de faux positifs) tout en maintenant un bon taux de détection (faible taux de faux négatifs). L'exactitude globale élevée confirme la capacité du modèle à distinguer efficacement les trafics légitimes des attaques DDoS.

Conclusion

En résumé, le modèle de détection d'intrusions DDoS proposé dans ce rapport démontre des performances prometteuses pour la protection des réseaux contre les attaques par déni de service distribué (DDoS). L'analyse des résultats d'évaluation a révélé que le modèle atteint une précision élevée dans l'identification des flux malveillants tout en maintenant un bon taux de détection, avec une exactitude globale de 98,84%. Ce modèle présente plusieurs avantages significatifs à savoir efficacité, adaptabilité et interprétabilité. Malgré ces résultats encourageants, des améliorations futures peuvent être envisagées.

Ce projet ouvre la voie à de futures recherches et développements dans le domaine de la détection des intrusions DDoS. L'exploration de nouvelles architectures de réseaux neuronaux, l'intégration de techniques d'apprentissage par transfert et l'utilisation de données en temps réel sont quelques-unes des pistes prometteuses à explorer pour améliorer davantage la performance et l'applicabilité des modèles de détection d'intrusions. La collaboration entre chercheurs, professionnels de la cybersécurité et acteurs de l'industrie sera essentielle pour continuer à faire progresser l'état de l'art dans la lutte contre les cybermenaces et protéger les réseaux contre les attaques DDoS de plus en plus sophistiquées.